Software Engineering & Human Interface

CSET 3600 – 901

~~Team 4~~

Team Laser Explosion

Ryan Rapini
Allyn Cheney
Edward Verhovitz
Bruce Willis

Project Proposal

FRIDAY, SEPTEMBER 21, 2012

Our group has decided to implement a battleship game for the course project. The battleship game will have options for single player vs. a computer or for two players to go head to head in an *exhilarating and intense* multiplayer battle of awesome. Players will take turns firing missiles at each other's immobile fleet of military vessels in what is one of the most visceral war-simulators ever created. There will of course be some interesting features that we will implement. Neither player will be able to see the other player's board of course because that would pretty much defeat the purpose of Battleship. The AI will have several difficulty levels ranging from Battleship Grand Master to bumbling oaf, giving singleplayer games a bit of depth and excitement as well. The game will also log all plays made by both players and allow for them to be played back later. When one person wins we will probably make the game sing them a song or show them a funny picture or something, too.

On the programming side, we plan to use Python as our language of choice. Python is versatile and powerful, and has some of the best documentation around in addition to an abundance of open-source libraries we can use to implement various features. Also the mascot is a snake and I really like snakes. Python allows us to program in a very flexible manner since it bundles its own live interpreter and compiles at runtime. This will allow for easy testing, and version control is trivial when no one has to worry about build environments or handling binaries. Python is also cross platform, which is nice, as it will allow us to program on whatever computers we have already. All of the code will be Object Oriented for Great Justice, using only the most rigid and proper of programming practices.

We plan to implement this as a GUI application with some of the most awesome graphics we can find. Adam Servis has already volunteered to handle all of our graphic design. What a nice guy. Hopefully we can be approved for this.

Ed is in charge of handling the main game mechanics, since he has already told me he is an avid Battleship enthusiast. He will be handling the rules of the game, i.e. what ships can be placed where and how to tell if you have hit or sunk a ship. This is his passion, we'll leave him to it. I will be in charge of the multiplayer aspect of the game, and the netcode. I like to think my vast knowledge of online games will make me a natural for programming networking infrastructure. I also read a book on networking once, so that should come in handy. Allyn will be in charge of the game's AI for single player, in all of its varying

complexities. He will also try to iron out any other bugs we come across along the way. He will also be in charge of composing the game's musical score and Original Sound Track.

Beyond this, we don't have a lot planned, but I'm sure we will make an awesome project!

Use Cases

FRIDAY, October 12, 2012

Our battleship project will use a wide variety of use cases. The game will start with a main menu, from which the user will have the choice of either StartGameVsAI or StartGameVsPlayer. Regardless of which choices is picked, the program will bring them to a game board via the SetupGameBoard use case, which will then bring each user to the UserSetsShips use case. From there, the game will run via the RunGame use case, periodically CheckingForWinner and CheckForHit and CheckForSunkShip

Use Case: StartGameVsAI

Actors: Game player

Goal: To start a single player game vs. computer

Preconditions: Game is at menu

Trigger: User decides to play game vs. computer

Scenario:

1. Game menu is displayed
2. User clicks start game vs. computer button
3. System starts game vs. computer

Exceptions:

a. n/a

Use Case: StartGameVsPlayer

Actors: Game player 1 and 2

Goal: To start a networked game for 2 players

Preconditions: Game is at menu

Trigger: User decides to play game vs. another player

Scenario:

1. Game menu is displayed
2. User clicks start network game button
3. System goes to menu asking for network IP of second player
4. Player enters an IP
5. Game connects users

Exceptions:

- User doesn't enter IP for second player
- User doesn't enter valid IP for second player

Use Case: SetupGameBoard

Actors: System

Goal: To setup the game board

Preconditions: Game loading

Trigger: User has picked either game vs. AI or another player

Scenario:

1. Game is loading
2. Game sets up arrays to be used as game board
3. Game goes to game board

Exceptions:

- n/a

Use Case: UserSetsShips

Actors: Game player

Goal: To have user set ships up on their board

Preconditions: Game is loaded

Trigger: Game board has loaded

Scenario:

1. Game board is setup and game loaded
2. User places ships on their board

Exceptions:

- User fails to place all ships on the board, a message should popup informing them to place all their ships.
- User places ships improperly (overlapping) or off the board.

Use Case: RunGame

Actors: System

Goal: To start playing the game

Preconditions: Game is loaded and ships are placed

Trigger: User has clicked button after placing all their ships

Scenario:

1. Game is loaded and ships are placed
2. Randomly one player is prompted to start the game
3. Users keep picking locations one at a time until one person's ships are sunk

Exceptions:

- User picks a location they have already picked
- User picks an invalid location
- User leaves the game unexpectedly

Use Case: CheckforWinner

Actors: System

Goal: To check for game winner

Preconditions: Game is loaded, ships are placed and game is running

Trigger: User checks for a spot looking for a hit or miss

Scenario:

1. Game is running
2. User clicks a spot looking for a hit
3. Afterwards game checks to see if a user has won

Exceptions:

- n/a

Use Case: CheckforHit

Actors: System

Goal: To check if user hit a ship

Preconditions: Game is running

Trigger: User checks a spot

Scenario:

1. Game is running
2. User clicks a spot looking for a hit or miss
3. System returns either hit or miss

Exceptions:

- User clicks a spot they have already checked

Use Case: CheckforSunkShip

Actors: System

Goal: To check if user sunk a ship

Preconditions: Game is running

Trigger: User checks a spot

Scenario:

1. Game is running
2. User clicks a spot looking for a hit or miss
3. System informs user if they sunk a ship

Exceptions:

- n/a

Use Case: CheckforMiss

Actors: System

Goal: To check if user missed a ship

Preconditions: Game is running

Trigger: User hits empty field of play

Scenario:

1. Game is running
2. User clicks a spot looking to hit a ship
3. System informs user of a miss

Exceptions:

- n/a

Project Plan

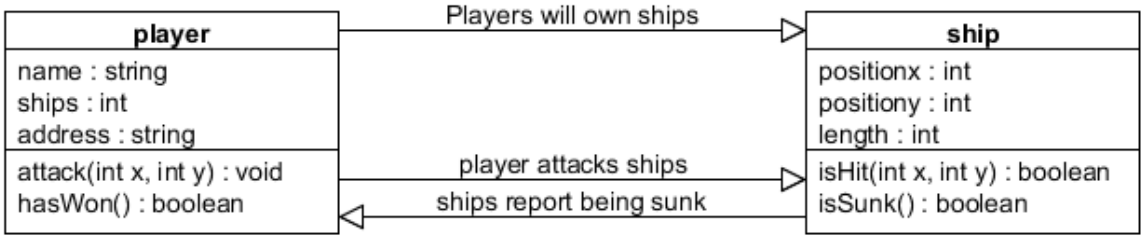
FRIDAY, OCTOBER 19, 2012

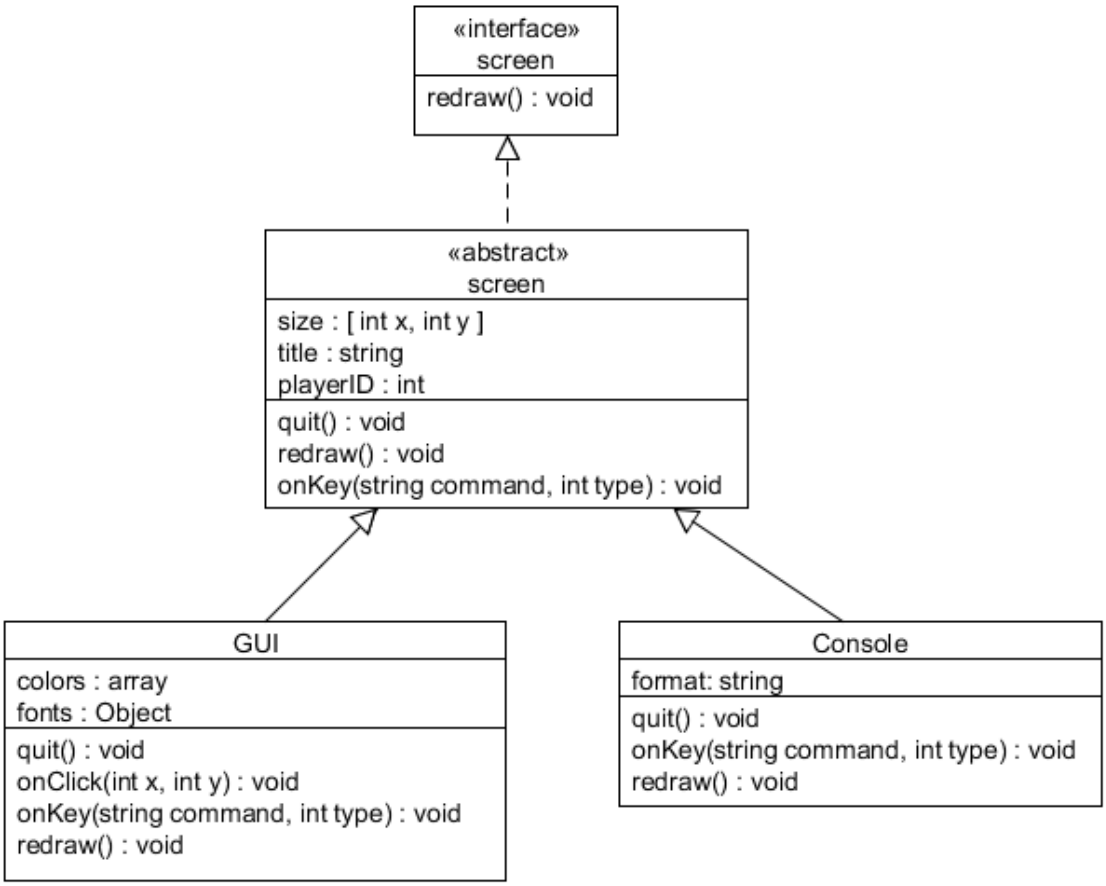
Start	End	Assigned To	Description
10/18/12	12/01/12	Allyn	Design board and game piece objects.
10/25/12	12/01/12	Allyn	Create AI for single player game.
11/1/12	12/01/12	Ryan	Create netcode for multiplayer game.
11/8/12	12/01/12	Allyn	Create code for various game conditions. Such as, hit, ship sunk, and game won.
11/1/12	12/01/12	Ryan	Create main program implementing above objects to create whole game. This would include the game run function.
10/18/12	12/01/12	Ryan	Design GUI for game.
10/25/12	12/01/12	Eddie	Design Graphics elements for GUI.
10/25/12	12/01/12	Eddie	Design Sound elements for game.
11/21/12	12/01/12	Ryan, Allyn, Eddie	Add GUI elements to main game and complete game.

UML Diagrams

FRIDAY, OCTOBER 26, 2012

We have several UML Diagrams, one implementing the interactions between players and their ships ingame, and the other implementing the interface between the screen and either the GUI or the Console interface.





Code Review 1

Friday, November 02, 2012

So far development has been moving along slowly but surely. We are writing our game in Python, making use of the pygame library to handle our GUI needs and doing most of the rest of the code by hand. We spent a bit of time setting up our personal development environments, and Allyn and Eddie have been slowly familiarizing themselves with Python, a language neither had worked in thus far. I have taken this time to start laying down the foundations for the work ahead of us, and also to provide feedback to the other two as they begin coding. I wrote a few bits of sample code for them to work with, and have also begun research for our netcode work later into the project.

As far as the main code goes, Allyn and I have made significant progress on the start of our GUI:

Aside from this, most of the progress has been in the form of structuring and layout for future code. Allyn has created several classes for our ships and for the game board, and I have been building GUI elements in preparation for the main menu. Eddie has been hard at work designing graphical elements for the front end of the GUI I am building, as he continues to acquaint himself with the syntax of Python.

We have been collaborating via email correspondence, via piratepad.net (an online collaborative typing tool) and via git (distributed version control system) hosted on github here: https://github.com/ryanrapini/CSET3600

Project Plan (Updated)

Friday, November 02, 2012

Start Date	End Date	Assigned To	Description	Status
10/18/12	12/01/12	Allyn	Design board and game piece objects.	Nearing Completion
10/25/12	12/01/12	Allyn	Create AI for single player game.	
11/1/12	12/01/12	Ryan	Create netcode for multiplayer game.	
11/8/12	12/01/12	Allyn	Create code for various game conditions. Such as, hit, ship sunk, and game won.	In Progress
10/26/12	12/01/12	Ryan	Create main program implementing above objects to create whole game. This would include the game run function.	In Progress
10/26/12	12/01/12	Ryan	Design GUI for game.	In Progress
10/25/12	12/01/12	Eddie	Design Graphics elements for GUI.	In Progress
10/25/12	12/01/12	Eddie	Design Sound elements for game.	
11/21/12	12/01/12	Ryan, Allyn, Eddie	Add GUI elements to main game and complete game.	
TBA	TBA	TBA	Logging all plays made by the user	
TBA	TBA	TBA	Adding the ability to “replay games”	

Project Plan (Numero 2)

Friday, November 9, 2012

Start	End	Assigned To	Description	% Complete	Date Completed
10/18/12	12/01/12	Allyn	Design board and game piece objects.	100%	11/01/12
10/25/12	12/01/12	Allyn	Create AI for single player game.	25%	
11/1/12	12/01/12	Ryan	Create netcode for multiplayer game.	0%	
11/8/12	12/01/12	Allyn	Create code for various game conditions. Such as, hit, ship sunk, and game won.	25%	
11/1/12	12/01/12	Ryan	Create main program implementing above objects to create whole game. This would include the game run function.	5%	
10/18/12	12/01/12	Ryan	Design GUI for game.	23.5%	
10/25/12	12/01/12	Eddie	Design Graphics elements for GUI.	99.45%	11/08/11
10/25/12	12/01/12	Eddie	Design Sound elements for game.	100%	11/03/12
11/21/12	12/01/12	Ryan, Allyn, Eddie	Add GUI elements to main game and complete game.	5%	

Project Plan (The Third)

Tuesday, November 27, 2012

Start	End	Assigned To	Description	% Complete	Date Completed
10/18/12	12/01/12	Allyn	Design board object	100%	11/01/12
10/25/12	12/01/12	Allyn	Create AI for single player game.	75% - Needs more intelligent AI and more varied playstyles	
11/1/12	12/01/12	Ryan	Create netcode for multiplayer game.	25% - Needs to work over the internet	
11/8/12	12/01/12	Allyn	Create code for various game conditions. Such as, hit, ship sunk, and game won.	100%	11/24/12
11/1/12	12/01/12	Allyn	Allyn: Implement single player code in main loop.	100%	11/24/12
10/18/12	12/01/12	Allyn	Design GUI for game.	100%	11/24/12
10/25/12	12/01/12	Eddie	Design Graphics elements for GUI.	100%	11/25/12
10/25/12	12/01/12	Eddie	Design Sound elements for game.	100%	11/25/12
11/28/12	12/01/12	Everyone	Bugtest!	In progress!	

Test Suite

Tuesday, November 27, 2012

Check for Win:

This test is to check the opponent's attack board for all sunken ships.

```
def checkforwin(board):
    winarray = [0,0,0,0,0,0,0,0]
    win = False
    for x in range(BOARDWIDTH):
        for y in range(BOARDHEIGHT):
            temp = board.returnpiece(x,y)
            winarray[temp] = winarray[temp] + 1
    temp2 = 2
    while(temp2 < 7):
        if winarray[temp2] == temp2:
            win = True
        else:
            win = False
            break
        temp2 = temp2 + 1
    return win
```

Check for Sunk Ship:

This test will check the board for any ship that has been completely sunk.

```
def checkforshipsunk(board, piece, screen):
    sunk = False
    hold = 0
    for x in range(BOARDWIDTH):
        for y in range(BOARDHEIGHT):
            if (board.returnpiece(x,y) == piece):
               hold = hold + 1
    if (hold == piece):
        if (piece == 6):
            printstatus(screen, 'You sunk my Aircraft Carrier!')
        elif (piece == 5):
            printstatus(screen, 'You sunk my Battleship!')
        elif (piece == 4):
            printstatus(screen, 'You sunk my Submarine!')
        elif (piece == 3):
            printstatus(screen, 'You sunk my Destroyer!')
        elif (piece == 2):
            printstatus(screen, 'You sunk my Patrol Boat!')
```

Check for Hit, Miss, and Win:

This bit of code is used to check for a hit, a miss and if the last ship has of the opponent's board has been sunk, winning the game. Nested If statements take care of all testing with else statements if found to not be true for each test.

```
if (place >= 5):
    if (turn == 0):
        if (boxx != None and boxy != None) and mouseClicked:
            place = place + 1
            temp = cpuboard.checkforhitormiss(boxx,boxy)
            if (temp == 9):
                blah = 0
            else:
                playerattackboard.setpiece(temp,boxx,boxy)
                if (temp == 7):
                    printstatus(screen, 'Miss')
                    miss.play(loops = 0)
                else:
                    printstatus(screen, 'Hit')
                    hit.play(loops = 0)
                if (checkforwin(playerattackboard)):
                    printstatus(screen, 'You win!')
                    turn = -1
                else:
                    checkforshipsunk(playerattackboard, temp, screen)
                    turn = 1
```

Initialize Pygame:

Pygame is a library of scripts that we use to accomplish most of the graphics and audio for our game. This test will try to initialize Pygame and notify us if it fails. If it cannot find pygame, the game will completely quit.

```
def init():
    # Initilize Pygame. THIS MUST HAPPEN BEFORE ANYTHING ELSE CAN TOUCH THE
      PYGAME UTILS.
    try:
        pygame.init()
    except Exception as ex:
        print("Failed to load pygame. Exception is:\n{0}".format(ex))
        # If pygame doesn't load, we might as well just give up.
        sys.exit(1)
    else:
        print("Pygame initilized sucessfully.")
```

Load Sound:

As with the previous test, this will attempt to load the sound libraries for the game and initialize the mixer with preset frequencies, channels, and bit rates. If it fails, we print a warning.

```
# Load sound
try:
    loadSound()
except Exception as ex:
    print("Failed to load sound. Exception is:\n{0}".format(ex))
else:
    print("Sound loaded sucessfully.")
```

Project Plan (Final Submission)

Friday, December 07, 2012

Start	End	Assigned To	Description	% Complete	Date Completed
10/18/12	12/01/12	Allyn	Design board object	100%	11/01/12
10/25/12	12/01/12	Allyn	Create AI for single player game.	100%	
11/1/12	12/01/12	Ryan	Create netcode for multiplayer game.	100%	
11/8/12	12/01/12	Allyn	Create code for various game conditions. Such as, hit, ship sunk, and game won.	100%	11/24/12
11/1/12	12/01/12	Allyn	Allyn: Implement single player code in main loop.	100%	11/24/12
10/18/12	12/01/12	Allyn	Design GUI for game.	100%	11/24/12
10/25/12	12/01/12	Eddie	Design Graphics elements for GUI.	100%	11/25/12
10/25/12	12/01/12	Eddie	Design Sound elements for game.	100%	11/25/12
11/28/12	12/01/12	Everyone	Bugtest!	All good!	